The Yael library

Matthijs Douze, Hervé Jégou

The Yael library

Matthijs Douze Hervé Jégou
Inria

Abstract

This paper introduces YAEL, a library implementing computationally intensive functions used in large scale image retrieval, such as neighbor search, clustering and inverted files. The library offers interfaces for C, Python and Matlab. Along with a brief tutorial, we analyze and discuss some implementation choices and their impact on efficiency. `http://yael.gforge.inria.fr/`

1. Motivation and overview

The motivation of YAEL is twofold: We aim at providing

- core and optimized instructions and methods commonly used for large-scale multimedia retrieval systems;
- more sophisticated functions associated with state-of-the-art methods, such as the Fisher vector [11, 10], VLAD [8, 10], Hamming Embedding [5, 6] or more generally methods based on inverted file systems, such as selective match kernels [13].

Having these two kinds of functions implemented in the core library, we can rapidly disseminate open source packages reproducing the results of recent papers for image retrieval. YAEL is intended as an API and does not implement a retrieval system in an integrated manner: only a few test programs are available for key tasks such as k-means. Yet this can be done on top of it with a few dozen lines of Matlab or Python code.

We provide one sample package relying on YAEL, which implements an image retrieval system based on the Fisher vector. Yet we mention that several of such packages are available online, each being associated with a particular task and publisehd method. Section 3 lists a few of these, along with download statistics.

Design choices. YAEL is designed to handle dense data in float, as it is primarily used for signal processing tasks where the quality of the representation is determined by the number of dimensions rather than the precision of the components. In the Matlab interface, single matrices, and float32 in Python.

YAEL was designed initially to manipulate matrices in C. It was interfaced for Python using SWIG, which gives low-level access to the full library. An additional Numpy layer (ynumpy) is provided for high-level functions. The most important functions of YAEL are wrapped in Mex to be callable from Matlab.

Performance is very important. YAEL has computed k-means with hundreds of thousand centroids and routinely manipulate matrices that occupy more than 1/2 the machine's RAM. This means it is lightweight and 64-bit clean.

The design choices of YAEL are governed by efficiency concerns more than by portability. The library is maintained for Linux and MacOS. Although an older version of the library was ported to Windows OS, only a few functions of the Matlab interface work on this platform.

Dependencies. YAEL relies on as few external libraries as possible. The only mandatory ones are BLAS/Lapack (for performance) and the Python/C API. Other libraries (Matlab's mex, Arpack, OpenMP) are optional.

Threading. Functions of significant complexity are parallelized. The level at which threading occurs must be chosen carefully: at the BLAS level, the YAEL function or the calling function. Therefore, YAEL functions often take a nt parameter, the number of threads.

Interface. YAEL contains hundreds of functions at the C level. Only a few are exported in Matlab and Python, as many basic matrix operations are natively available in these environments. In C, the functions are prefixed with the type of data they operate on. For example, fvec_max takes a fvec, or "float vector" array as input. In Matlab, most of the functions are prefixed with yael_. In Python, YAEL is available via the ynumpy module, which provides a sufficient context.

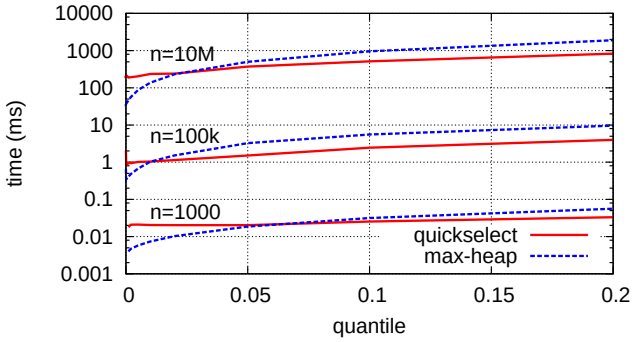

Figure 1. Runtime of the quickselect and max-heap methods to find a quantile in a table of uniformly distributed scalars (one computing core).

The rest of this paper is organized as follows. Section 2 lists the main functions, and shows on a few representative cases how we have optimized the functions. Section 3 shows the current impact of Yael with respect to various indicators: number of downloads, derived packages and research papers referring to the URL library.

2. Main functions

Here we review the implementation of YAEL's main functions, with their names in C. Table 1 gives the Python and Matlab equivalents of these functions.

2.1. k-smallest values query

YAEL contains two implementations of the algorithm to find the k smallest values of a table of scalars:

- The quick-select algorithm (a.k.a. Hoare method). It amounts to applying a quick-sort on the table, but without handling the fraction of the table that does not contain the quantile limit.
- The max-heap algorithm, that maintains the set of k smallest values seen so far in a max-heap. The max-heap is a binary tree optimized to perform the operations "remove the largest value" and "insert a new value", both of which having complexity $\log(k)$.

Figure 1 shows that for smaller quantiles, the max-heap is faster. The function fvec_k_min automatically switches between the two versions, depending on the parameter k and number of elements.

2.2. Searching neighbors

Image retrieval requires finding similar elements on large scale. YAEL implements exact algorithms to retrieve nearest neighbors. For larger scale, one may resort to indexing structures such as those proposed in the inverted file structure of YAEL (see Subsection 2.6).

Exhaustively searching for nearest neighbors requires to compute the distance matrix $D \in \mathbb{R}^{m\times n}$ from the query vectors $X \in \mathbb{R}^{d\times n}$ to the database vectors $Y \in \mathbb{R}^{d\times m}$ (in the Euclidean case). Then, the smallest value in each column j of D is selected: it corresponds to the the nearest neighbor of query j. Both computations are performed in blocks to improve cache locality and because D may not fit in memory.

Euclidean space. We exploit the equivalence:

$$D = S(X)1_n^\top + 1_m^\top S(Y) - 2X^\top Y \qquad (1)$$

where 1_n is a $n-$vector of ones and $S(X)$ is the vector of squared norms of the columns X. The costly operation is the $X^\top Y$, for which we use the BLAS sgemm. When more than one nearest neighbor is required, we use a max-heap over the columns of D, as discussed in the previous subsection.

Hamming space. To compute Hamming distances, we employ the `xor` and SSE4.2 popcnt instructions: the Hamming distance between two 64-bit integers is computed in two processor cycles. Hamming distances are typically employed to retrieve all vectors within a given distance to the query. Therefore, unlike in the k-NN case, the list of neighbors must be adjusted during the query.

Other distances. The cosine similarity is optimized with matrix multiplication and cache optimization. The ℓ_1 and χ^2 distances are optimized using specific SSE instructions.

2.3. Principal Component Analysis

Dimensionality reduction with PCA is widely used. However, many implementations are not tractable for a large number of high-dimensional vectors,. YAEL provides distinct implementations for Matlab and C. The Matlab function yael_pca automatically switches between the Gram and the covariance methods [2]. Hereafter, we detail the variants implemented in the C interface, which include both these variants and tricks that limits the memory consumption.

Computing the PCA of a set of vectors $X = [x_1 \dots x_n] \in \mathbb{R}^{d\times n}$ proceeds as follows.

- centering the data:

$$\bar{X} = X - \bar{x}1_n^\top \quad \text{with} \quad \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad (2)$$

- computing the SVD decomposition of $\bar{X}$: $\bar{X} = U\Sigma V^\top$ where $U \in \mathbb{R}^{d\times d}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{d\times n}$ is positive diagonal, padded with zeros on the right or bottom, with elements $[\sigma_1^2, \dots, \sigma^2_{\min(d,n)}]$.The singular values are sorted: $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min(d,n)}$.

Matlab function	Python function	Description
yael_kmax		Find the k largest elements of a vector (or set of vectors)
yael_kmin		Find the k smallest elements of a vector (or set of vectors)
yael_cross_distances	cross_distances	Compute all distances between two sets of vectors
yael_hamming		Compute all Hamming distances between two sets of packed bit-vectors
yael_nn	knn	Find the nearest neighbors in a set for various distances
yael_kmeans	kmeans	Learn a k-means quantizer from a set of input vectors
yael_gmm	gmm_learn	Estimate a GMM from an input set of vectors
yael_fisher	fisher	Compute the GMM- Fisher vector associated with a given set
yael_ivf		Handle (construct, query, etc) an inverted file structure

Table 1. Main functions in the Matlab interfaces of YAEL, and corresponding Python function.

Given this decomposition, the PCA reduction of a vector $x \in \mathbb{R}^d$ (not necessarily from the initial $\{x_1 \dots x_n\}$) is $x' = U^\top(x - \bar{x})$. Its main property is that, if vectors are drawn from a multidimensional Gaussian, the subvector x'_k of the k first components of x' is as close as possible to x'. Therefore, the PCA is often used to reduce the dimensionality from $\mathbb{R}^d$ to $\mathbb{R}^k$.

The only data we need to perform dimensionality reduction is $\bar{x}$ and the first k columns of U. Thus, the complete SVD is not necessary. There are two cases:

- Most often, $n > d$, *i.e.*, we have enough training data. In this case only the eigenvalues σ_i^2 and vectors U for the empirical covariance matrix $\bar{X}\bar{X}^\top$ are computed. The cost of computing these eigenvectors is $9d^3$ flops [3, algorithm 8.3.3].

- When $d > n$, it is more efficient [2] to compute the eigenvectors V of the Gram matrix $\bar{X}^\top\bar{X}$. Denoting $(u_i)_{i=1\dots d}$ and $(v_j)_{i=1\dots n}$ the columns of U and V respectively, $u_i = 1/\sigma_i^2 X v_i$ for $i = 1 \dots n$. The remaining columns can be obtained by completing the orthonormal basis.

Finding eigenvalues in YAEL uses the Lapack ssyev function. When the data matrix X is too large to fit in RAM, the mean $\bar{x}$ and covariance $\bar{X}\bar{X}^\top$ are computed in a single pass over the matrix, using YAEL's pca_online_* functions.

Partial PCA. When the number of required dimensions $k \ll d$, it is possible to efficiently compute only part of the eigenvalues and eigenvectors. This can be done either by:

- reducing the matrix to tridiagonal form, then only partially diagonalizing this representation (Lapack function ssyevx).

- using an iterative Lanczos method [3, section 9.1] to compute the required components. The advantage is that the covariance matrix does not need to be computed, because it is used only to perform matrix-vector operations $\bar{X}\bar{X}^\top v$ for a given $v \in \mathbb{R}^d$. YAEL uses the Arpack ssaupd/sseupd functions.

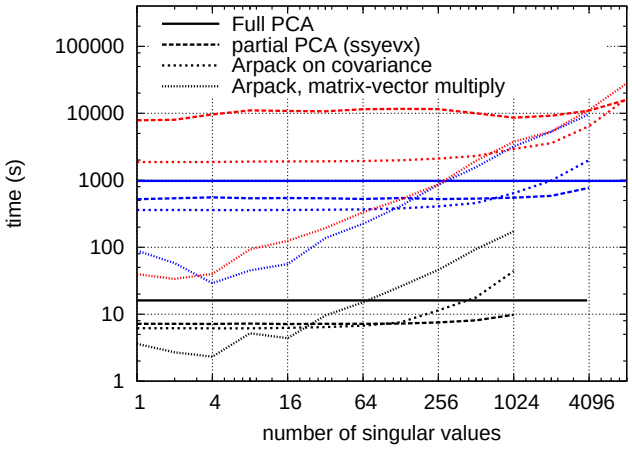

Figure 2. Cost of computing partial PCAs using different methods, on matrices of various sizes (on a 40-core computer): black = 4096×40000, blue = 16384×160000, red = 65536×88000. *Best viewed in color.*

Figure 2 shows the cost of computing a partial PCA. The on-the-fly multiplication is more efficient when k is small because the covariance matrix is not computed. For larger k, Arpack is faster at first, then Lapack's tridiagonalization.

These functions are implemented in fmat_new_pca_part and pca_online_complete_part functions.

2.4. k-means Clustering

The cost of the k-means clustering is determined by that of the nearest-neighbor search (see Section 2.2). Despite its simplicity, the performance and stability of k-means depend on some implementation details:

- initialization: YAEL can use a random or the kmeans++-style initialization [1]. We observed, however, that for a large number of centroids, the random initialization is as good, and it is much faster because kmeans++ cannot be parallelized. The user can also provide his own initialization for the centroids.

- vanishing clusters: sometimes no more points are assigned to a cluster. This happens even on "well-behaved" datasets, for example when training product quantization [7] vocabularies on very large vectors that may be sparse. In this case, YAEL randomly chooses another cluster and splits it in two, in a way that avoids a new cluster to vanish at the next iteration. This a done by selecting a big cluster associated with centroid c. Two clusters are obtained from it as $c - \varepsilon$ and $c + \varepsilon$, where ε is a small random perturbation vector.

- restarting: kmeans computes the solution to an optimization problem using an iterative method that converges to a local minimum. The YAEL kmeans can perform this optimization starting from different initial points and select the best one.

Several functions are available for k-means in the C interface of YAEL, depending on the desired output (average error per cluster, assignment, etc) and initialization options. In Matlab, k-means is called with function yael_kmeans, see help yael_kmeans for possible options. For instance, a simple call is done by

```
[C, I] = yael_kmeans (v, k, 'niter', niter);
```

where C is a matrix containing the k clusters, and I indicates to which cluster each input vector v is assigned.

2.5. Fisher vectors

The YAEL Gaussian Mixture Model [2, Section 2.3.9] estimation is initialized from a kmeans vocabulary of the same size, then expectation-maximization steps are performed. A set of vectors can be aggregated to a Fisher vector [11]. This is particularly useful to generate an image representation from local descriptors.

The two functions are implemented in YAEL's gmm_learn and gmm_fisher. A toy Matlab package implementing a baseline based on Fisher is available online, see Section 3. A similar Python package is provided for the open-source competition to illustrate YAEL's capabilities.

2.6. Inverted files

Inverted file systems [14] are a key component of existing large-scale image and video retrieval methods. For image search, they are widely adopted to implement the similarity computation for the methods derived from the seminal Video-Google paper [12], where sparse bag-of-words vectors are to be compared in an efficient way.

Although an inverted file system can be implemented by a sparse matrix-vector multiplication, a devoted implementation can be more efficient. Furthermore, more precise image search systems have been subsequently proposed, such as Hamming Embedding [6], selective match kernel [13] or nearest neighbor search with product quantization [7]. These strategies add some per-descriptor information, given in a form of a short code, *e.g.*, a binary vector.

YAEL implements a generic inverted system capable of storing and exploiting such additional information in the retrieval phase. It can be used with any quantizer, since the quantization is done externally. For instance, in the Matlab interface, one can learn and produce a structure for Hamming Embedding by providing a function handler for the quantizer (exact Euclidean assignment in the example below) when calling the yael_ivf_he function:

```
>> quantizer=@yael_nn;
>> qparams=yael_kmeans(xtrain, k,'niter',20);
>> ivfhe=yael_ivf_he (k,nbits,x,quantizer,qparams);
```

The entry point for all operations is the function yael_ivf: adding new elements, performing queries, saving and loading an inverted file, etc. A full example is given in the subdirectory `test/matlab` of the YAEL package.

3. Impact of Yael

This section gives a brief overview of the impact of YAEL, both in terms of packages that depend on it, and with respect to usage statistics measured by the number of downloads. In addition, we mention that about 20 papers or reports refer to YAEL, as indicated by the results of the Google scholar query "gforge inria yael".

3.1. Derived software

Several software use YAEL as a core library, in particular online packages that reproduce the results of papers[1]. Hereafter, we describe a few packages developed at Inria.

VLAD and Fisher – Image representations. The packages have been released to reproduce the results of the corresponding papers [8, 10]. They are available online on a dedicated webpage[2]. The Fisher implementation reproduces the results of the original authors at Xerox Research Center Europe.

Product quantization [7] is a state-of-the-art method for approximate nearest neighbor search in the compressed domain. A commercial version of this algorithm is proposed by Inria. Alternately, an open-source toy package in Matlab is available online[3], and depends on YAEL. Another package implementing approximate search with short codes implements the **anti-sparse** binarization technique [9] is derived from YAEL[4].

[1] http://people.rennes.inria.fr/Herve.Jegou/software.html
[2] http://lear.inrialpes.fr/src/inria_fisher
[3] http://people.rennes.inria.fr/Herve.Jegou/projects/ann.html
[4] https://gforge.inria.fr/projects/antisparse

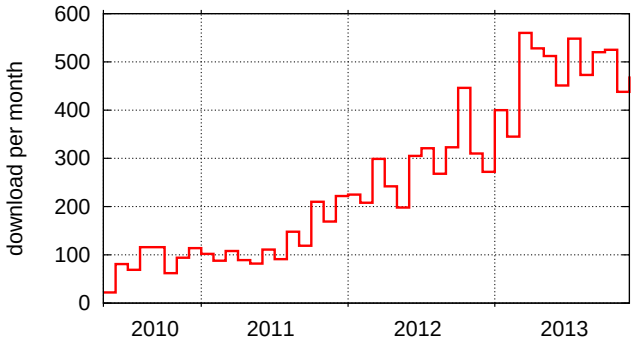

Figure 3. Statistics on YAEL files downloads. These measures include all files (including documentation files and related packages), but not the SVN checkouts or updates by developers and privileged users at Inria.

Babaz – Audio similarity search. Babaz [4] is an audio search engine[5] for video copy detection based on the audio track. The library is also usable in applications like Shazaam, however in this context there are faster and as accurate methods specifically devoted to music search. The method implemented has been used for Trecvid'2011, where it obtained the best results for the audio part.

Selective match kernels [13] are a state-of-the-art method for image search, with a focus on best ranking performance. It relies on YAEL's inverted file. A full package reproducing the paper's results is available online in the download section[6] of YAEL.

3.2. Download statistics

YAEL is released on Inria's gforge, a facility similar to that provided by Sourceforge. This offers several facilities like trackers and forums. It also produces usage statistics. For instance, Figure 3 reports the number of downloads per month between April 2010 (first release) and December 2013. The last version v371 and related packages have been downloaded 2200 times in the last five months (between December 2013 and May 2014), 1275 of these downloads corresponding to the core library. The library clearly has an increasing success, presumably because of the increasing number of packages relying on it.

As a final note, several demonstrators, in-house code and commercial software employ YAEL as a core library. YAEL is used in the Inria bigimbaz image search engine since 2008, which indexes over 10 millions images, to perform efficient image queries submitted by the users on the online interface[7]. YAEL was used in the winning submissions to Trecvid Multimedia Event Detection in 2012 and 2013.

[5]http://babaz.gforge.inria.fr/

[6]https://gforge.inria.fr/frs/?group_id=2151

[7]http://bigimbaz.inrialpes.fr

4. Conclusion

YAEL is a library which offers computationally intensive functions that are critical for large scale image search. Its main purpose is to facilitate the reproducibility of results, thanks to release of packages associated with research papers. Yael offers C, python and Matlab interfaces for Linux and MacOS operating systems. The core functions are heavily optimized with SSE operations and calls to Lapack/Blas matrix libraries. The development process follows the motto "release soon, release often".

Acknowledgements. The library was first developed in the context of the Quaero program. It is currently supported by ERC VIAMASS no. 336054.

References

[1] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *ACM-SIAM symposium on Discrete algorithms*, 2007.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[3] G. H. Golub and C. F. V. Loan. *Matrix Computations*. John Hopkins University Press, Baltimore and London, second edition, 1991.

[4] H. Jégou, J. Delhumeau, J. Yuan, G. Gravier, and P. Gros. Babaz: a large scale audio search system for video copy detection. In *ICASSP*, Mar. 2012.

[5] H. Jégou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, Oct. 2008.

[6] H. Jégou, M. Douze, and C. Schmid. Improving bag-of-features for large scale image search. *IJCV*, 87(3):316–336, Feb. 2010.

[7] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *Trans. PAMI*, 33(1):117–128, Jan. 2011.

[8] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010.

[9] H. Jégou, T. Furon, and J.-J. Fuchs. Anti-sparse coding for approximate nearest neighbor search. In *ICASSP*, Mar. 2012.

[10] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid. Aggregating local descriptors into compact codes. In *Trans. PAMI*, 2012.

[11] F. Perronnin and C. R. Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*, 2007.

[12] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *ICCV*, 2003.

[13] G. Tolias, Y. Avrithis, and H. Jégou. To aggregate or not to aggregate: Selective match kernels for image search. In *ICCV*, 2013.

[14] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):6, 2006.